System Requirements Specification

(*Pflichtenheft*)

(TINF18C, SWE I Praxisprojekt 2019/2020)

Project: AMLEngine-DLL Interfaces

Customer: Rentschler & Ewertz
Rotebühlplatz 41
70178 Stuttgart

Supplier: Team 4: Joshua, Kevin, Krister, Lucas, Markus, Robin
Rotebühlplatz 41
70178 Stuttgart

Version	Date	Author	Comment
0.1	21.10.2019	Lucas	Document created
0.2	01.10.2019	Lucas	Added nonfunctional requirements
1.0	04.11.2019	Lucas	Document finished
1.5	01.05.20	Lucas	Updated data
2.0	04.05.20	Lucas	Updated and Completed

Table of Content

1. Introduction

The goal of this project is to develop software which facilitates the usage of the AMLEngine[1] in the programming languages C++ and JavaScript.
In the case of JavaScript to achieve this, the functionality should be provided in the form of code, which can be used in other projects to interface with the AML.Engine.dll. For C++ however, the wrapper is a collection of instructions and examples on how to use the AMLEngine not the creation of code with which other software interfaces.
The application also provides a console application tool, which validates AML files and (de)compresses AMLX files.

1.1. Product Environment

The wrappers will be used in environments where users work with AML[2] files in programming languages other than C#. The source code aims to provide information on how the functionality can be used in JavaScript and C++.
The JavaScript wrapper can be used in projects for the JavaScript runtime environment Node.js. Node.js is a JavaScript runtime which is used to build scalable applications in JavaScript [3].

Other parts of the software can be used to validate AML files with a console application. It is also possible to unpack or pack AMLX files.

1.2. Use Cases

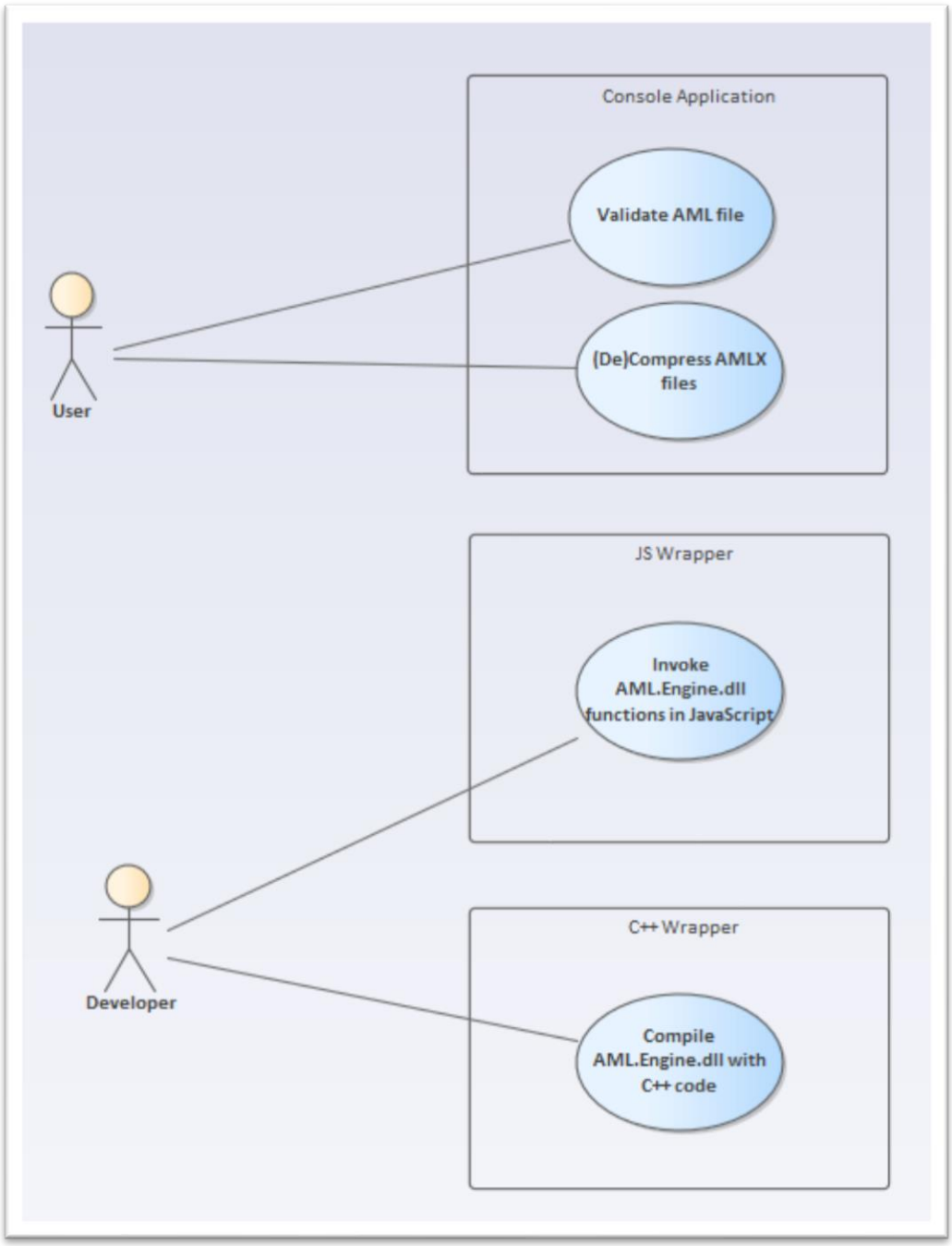

Figure 1: Use Cases

1.2.1. <UC.001> C++ Wrapper

Related Business Process:	<BP.001>: C++ Wrapper
Use Cases Objective:	Developer building a C++ project which uses the AML.Engine.dll with the help of the C++ wrapper instructions.
System Boundary:	Project boundaries
Precondition:	A Windows computer with an internet connection must be available.
Postcondition on success:	It is possible to develop a C++ program, which uses the AML.Engine.dll.
Involved Users:	AML.Engine.dll, wrapper, developer
Triggering Event:	Developer wants to import the AML.Engine.dll in their C++ project.

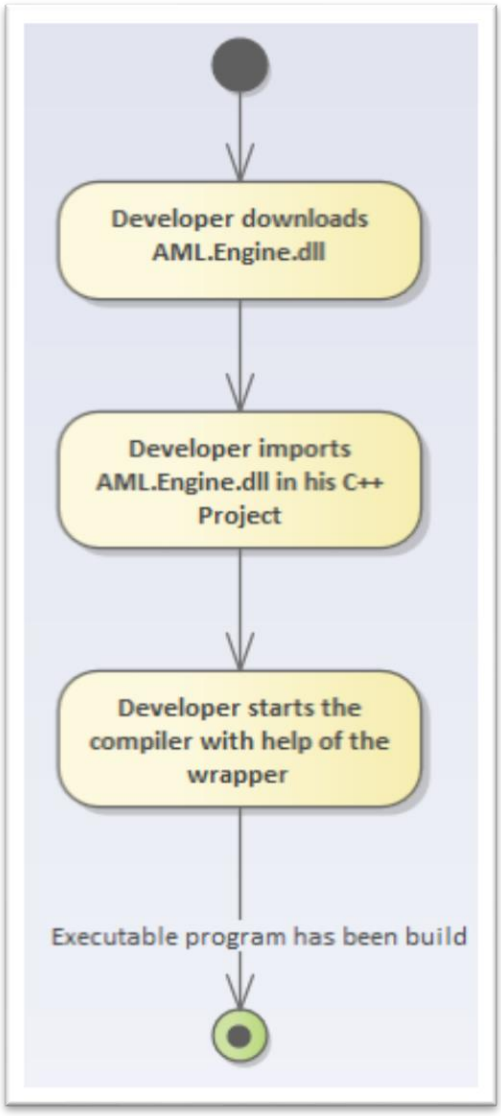

Figure 2: activity diagram C++ wrapper

1.2.2. <UC.002> JavaScript Wrapper

Related Business Process:	<BP.002>: JavaScript wrapper
Use Cases Objective:	Developer wants to use the functionalities from the AML.Engine.dll in a Javascript Node project.
System Boundary:	Project boundaries

Precondition:	The AML.Engine.dll must be available, wrapper must be imported into the project.
Postcondition on success:	It is possible to run the Node project, which uses the AML.Engine.dll
Involved Users:	AML.Engine.dll, wrapper, developer
Triggering Event:	Developer wants to import the AML.Engine.dll in their Node project.

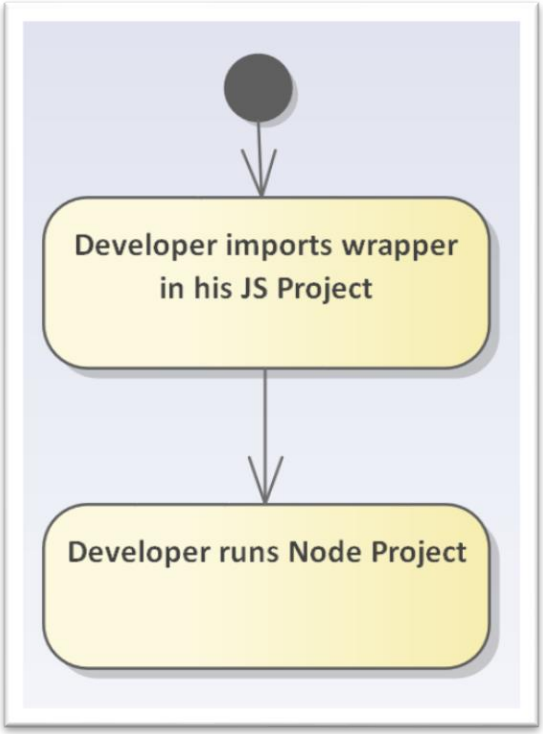

Figure 3: Activity diagram JavaScript wrapper

1.2.3. <UC.003> Validation in a Console Application

Related Business Process:	<BP.003>: Console Application
Use Cases Objective:	With help of a console application an AML file will be validated.
System Boundary:	console application
Precondition:	Program must be installed correctly.
Postcondition on success:	In case of failure, a helpful error message will be printed in the terminal. It also will be asked as whether one wants to repair the error.
Involved Users:	AML/AMLX file, console application, user
Triggering Event:	User wants to validate AML file

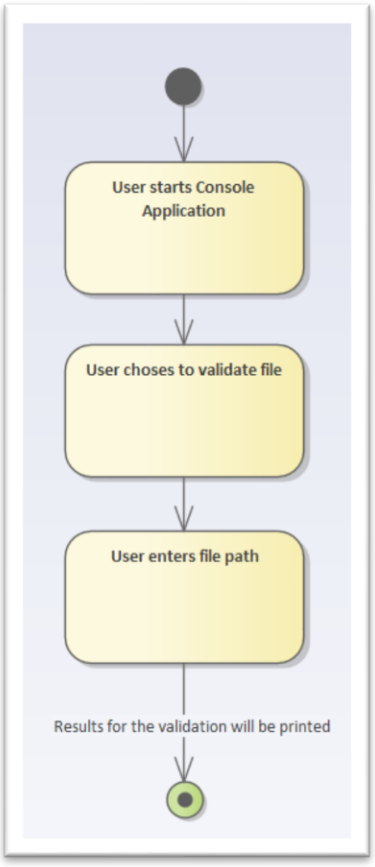

Figure 4: Activity diagram console application

1.2.4. <UC.004> Packing and Unpacking in a Console Application

Related Business Process:	<BP.003>: Console Application
Use Cases Objective:	With the help of a console application multiple AML files will be packed into an AMLX file or an AMLX file will be unpacked into AML files.
System Boundary:	console application
Precondition:	Program must be downloaded to the system. AML files and AMLX file must be valid.

Postcondition on success:	Files are packed or unpacked correctly.
Involved Users:	AML/AMLX file, console application, user
Triggering Event:	User wants to pack or unpack AML files.

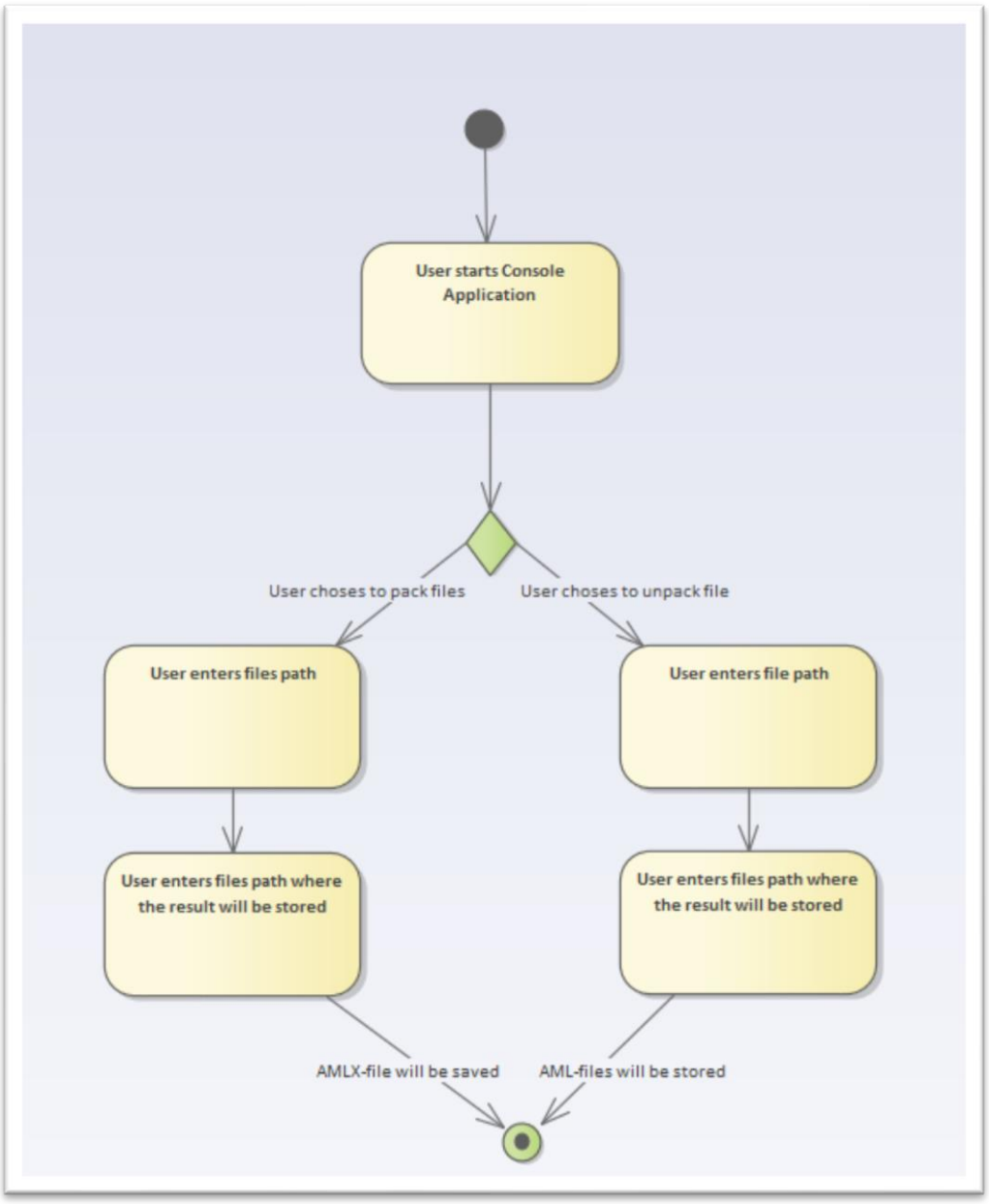

Figure 5: Activity diagram console application (un-)packing

2. Product Requirements

The following functionalities shall be supported by the system.

2.1. /LF10/C++ Functions

The C++ wrapper should enable developers to use all the functions of the AML.Engine.dll in their C++ project.

Input field	**Value Range**
AML.Engine.dll	Library to import and use

2.2. /LF20/C++ usability

It should be possible to compile the C++ project using the Microsoft C++ compiler and (optionally) the GNU compiler [5].

Input field	**Value Range**
C++ source code	The source code which must be compiled

2.3. /LF30/Javascript Functions

The JavaScript wrapper should enable developers to use most of the functions of the AML.Engine.dll in their Node.js project.

Input field	**Value Range**
JavaScript wrapper	The JavaScript wrapper which uses the AML.Engine.dll

2.4. /LF40/Javascript Usability

The project shall be able to be run in the JavaScript runtime environment Node.js.

Input field	**Value Range**
JS source code	The source code which should run in Node.js

2.5. /LF50/Import

The console application shall be able to read AML Files and import the important data from it. The Files will be provided by the User through a PATH parameter. Then the Application should access the File and extract the Information.

Input field	**Value Range**
PATH	Location of AML file

2.6. /LF60/(De-)Compression

The console application shall be able to Decompress AMLX files, which would be provided by the User. After that the Application imports the data from the extracted directory.
The console application shall be able to compress AML files, which would be provided by the User and save it to a specified directory.

Input field	**Value Range**
PATH	Location of AMLX file or AML files
DIRECTORY PATH	Location where the result should be stored

2.7. /LF70/Validation

The console application shall be able to parse and validate AML files. If it detects any errors in the AML file, it will print out the line number the error occurred on and the error itself. The information, given by the program, should help the User to fix the validation problem within an appropriate time range. It also will be asked as whether one wants to repair the error, if it is possible within the AMLEngine.

Input field	**Value Range**
PATH	Location of AML file

3. Product Data

3.1. /LD10/AML file

The Systems should be able to use and import AML files by specifying the file path. The valid path to these files will be provided by the user.

3.2. /LD20/AMLX file

The console application should read and decompress AMLX files into AML files. The console application should be able to compress multiple AML, and AML related, files into AMLX files.

4. Non-Functional Requirements

4.1. /NF10/Documentation

The documentation for each wrapper shall contain enough information for a developer to be able to start using the wrapper in their project. This includes a short introduction into the usage of the wrapper as well as some code samples. All supported functions and troubleshooting information will be available in a wiki.

4.2. /NF20/Installation

The console application should be able to run without any installation.

4.3. /NF30/Console Application Usability (optional)

The console application shall provide an easy to understand user interface which makes simple to select the desired options and parameters.

4.4. /NF40/Files

The console application shall support AML files and AMLX files. The wrappers shall support AML files.

4.5. /NF50/System Environment

The wrapper shall run under the Windows 7 operating system and newer windows versions.

4.6. /NF60/Dependencies

The library should have no external dependencies besides the .NET Framework or Node.js respectively and the AML.Engine.dll.

4.7. /NF70/License

The software shall be published under the MIT license [6].

5. References

[1] AMLEngine - https://github.com/AutomationML/AMLEngine2.1/

[2] AutomationML consortium: Whitepaper AutomationML. Part 1 - Architecture and general requirements.

[3] JavaScript runtime environment Node.js - https://nodejs.org/

[4] XML - https://www.w3.org/TR/xml/

[5] CAEX - https://www.plt.rwth-aachen.de/cms/PLT/Forschung/Projekte2/~ejwy/CAEX_IEC_62424/

[5] GCC/GNU - https://gcc.gnu.org/

[6] MIT license - https://choosealicense.com/licenses/mit/

6. Glossary

JS	JavaScript is a scripting language which supports dynamic typing.
AML	Automation Markup Language is an open standard data format for storing and exchanging plant planning data.
AMLX	Multiple files can be stored compressed in an AML-Container (.amlx).
C++	C++ is an extension of the C programming language.
CAEX	Computer Aided Engineering Exchange
CLI	The Console Application Interface from Microsoft Windows.
DLL	Dynamic Link Library is a file format used to store precompiled code
GUI	Graphical User Interface
.NET	The .NET Framework is a software development and runtime developed by Microsoft for Microsoft Windows.
Runtime environment	A runtime environment is an environment provided by the operating system for granting access to other system resources such as RAM.